CableLabs® Specifications
OnLine Content Access

Authentication and Authorization through Client Applications

CL-SP-AUTH-APP-I01-120118

ISSUED

Notice

This OLCA specification is the result of a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the cable industry and its customers. This document may contain references to other documents not owned or controlled by CableLabs. Use and understanding of this document may require access to such other documents. Designing, manufacturing, distributing, using, selling, or servicing products, or providing services, based on this document may require intellectual property licenses from third parties for technology referenced in this document.

Neither CableLabs nor any member company is responsible to any party for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document, or any document referenced herein. This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein. Distribution of this document is restricted pursuant to the terms of separate access agreements negotiated with each of the parties to whom this document has been furnished.

Document Status Sheet

Document Control Number:	CL-SP-AUTH-APP-I01-120118			
Document Title:	Authentication and Authorization through Client Applications			
Revision History:	I01 – Released 1/18/12			
Date:	January 18, 2012			
Status:	~~Work in Progress~~	~~Draft~~	**Issued**	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	**Public**

Key to Document Status Codes

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

Contents

Figures

Tables

1 SCOPE

1.1 Introduction and Purpose

The CableLabs OLCA Authentication and Authorization specification [AUTH1.0] enables streamlined implementation of access to video services across MSOs and Programmers. That specification primarily focuses on browser based use cases.

This document addresses access to video services through an 'application'. An application in this document is primarily treated as running on a mobile device (like an iPad or iOS/Android based tablet/phone). Access to video services through an application introduces challenges for security (that are implicitly addressed in case of a browser). These challenges include, among others, identity of the application accessing the content and user authentication. One requirement on the solution is also to take advantage of the previously defined OLCA infrastructure as much as possible.

The goal of this specification is to streamline the security interfaces between applications and video service providers (MSOs, Programmers, and Content Providers). Such standardization enables development of applications that can work seamlessly across all video service providers.

1.2 Eco-System

Figure 1 captures the eco-system addressed by this specification. These entities are explained later in Sections 4.1 and 4.2.

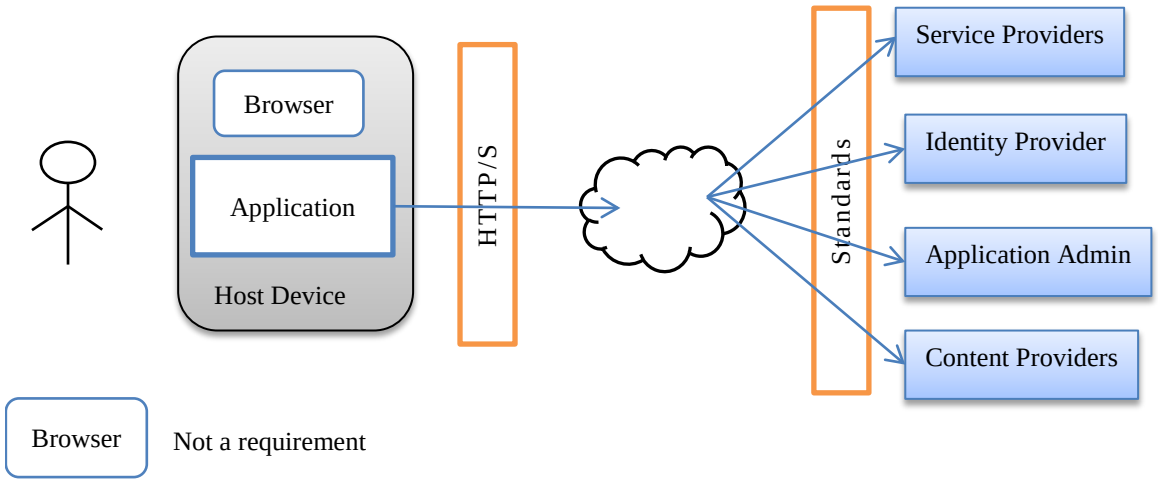


Figure 1 - Primary Entities in the Eco-system

1.3 Scenario Overview

Overall, the user experience is captured in the following flow.

1. User 'discovers' an application, downloads, and installs it. How this happens is outside the scope of this document.
2. User starts the application. Application may provide some initial content without any user authentication. How the application accesses such content is outside the scope of this document.
3. At some point, the service provider may require that the application can only be accessed from an authorized application. At that point, the application contacts the 'Application Admin', obtains an application token, and presents it to the service provider. This document defines the interfaces for fetching and passing the application token, and the format for the application token.

4. Service providers may also want to know what privileges an application carries so that they can be certain that a particular service can be provided to a particular application. This specification defines how the Application Admin can pass application's privileges in the application token.
5. At some other point, service provider may require the user be authenticated to access particular content/service. At that point, the application will use the flows defined in this specification to authenticate the user and pass the result of the authentication to service provider. This specification defines how the user can be authenticated at an MVPD by the application and how the result of that can be communicated to the service provider.

This specification utilizes the authentication and authorization interfaces defined [AUTH1.0] for user authentication and content authorization.

1.4 Assumptions

In determining the requirements and solution in this specification, the following assumptions are used as driving principles.

1. Applications may be developed by anyone in the eco-system – MSOs, Programmers, or Third-Party developers.
2. All services exposed by the MSOs or Programmers are accessed through Web Services standards (more specifically, HTTP/S based RESTful services).

1.5 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

All references are subject to revision, and parties to agreement based on this specification are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

[AUTH1.0] Authentication and Authorization Interface 1.0 Specification, CL-SP-AUTH1.0-I03-120118, January 18, 2012, Cable Television Laboratories, Inc.

[ID-BearerTokens] IETF Internet Draft, The OAuth 2.0 Authorization Protocol: Bearer Tokens, M. Jones, D. Hardt, and D. Recordon, draft-ietf-oauth-v2-bearer-15.txt, December 2011.

[ID-JWT] IETF Internet Draft, JSON Web Token (JWT), M. Jones, D. Balfanz, J. Bradley, Y. Goland, J. Panzer, N. Sakimura, P. Tarjan, draft-jones-json-web-token-07.txt, December 2011.

[ID-OAuth Assertions] IETF Internet Draft, OAuth 2.0 Assertion Profile, M. Jones, B. Campbell, Y. Goland, draft-ietf-oauth-assertions-01, October 2011.

[ID-OAuth] IETF Internet Draft, The OAuth 2.0 Authorization Protocol, E.H-Lahav, D. Recordon, D. Hardt, draft-ietf-oauth-v2-22.txt, September 2011.

[SAML Bindings] OASIS Standard, Bindings for the OASIS Security Assertion Markup Language (SAML) v2.0, March 2005, http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf.

[SAML Core] OASIS Standard, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0, March 2005, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

2.2 Reference Acquisition

- Internet Engineering Task Force (IETF) Secretariat, 48377 Fremont Blvd., Suite 117, Fremont, California 94538, USA, Phone: +1-510-492-4080, Fax: +1-510-492-4001, http://www.ietf.org
- Internet Engineering Task Force (IETF), Internet: http://www.ietf.org/
  Note: Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time.
  The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.
  Internet-Drafts may also be accessed at http://tools.ietf.org/html/
- OASIS Standards, http://docs.oasis-open.org/security/saml/v2.0

3 DEFINITIONS AND ABBREVIATIONS

3.1 Terms and Definitions

This specification uses the following terms:

Account	A key to the subscriber data at the MVPD. There can be multiple Subscribers belonging to the same account. A particular subscriber may be designated as the actual owner.
Application	Software that can be installed on mobile device.
Application ID	A unique identifier given to each App. This is opaque except to the App and the App Admin.
Assertion	A piece of data produced by a SAML authority regarding either an act of authentication performed on a subject, attribute information about the subject, or authorization data applying to the subject with respect to a specified resource.
Authentication	To confirm a system entity's asserted principal identity with a specified, or understood, level of confidence.
Authorization Server	An OAuth role. Responsibilities include support for Authorization Requests/Responses, initiating [SAML-based] user authentication.
Back-channel	A direct connection between two entities (not going through end user's device).
Client	Analogous to Application (note that 'Client' is an OAuth terminology). Also note that Client is NOT the user/subscriber.
Identity Provider	An entity that can verify user's credentials, can issue Assertions regarding successful user authentication, can provide relevant attributes about user's subscription and verify user's authorization to specific content.
Security Assertion Markup Language	A set of specifications describing security assertions that are encoded in XML, profiles for attaching the assertions to various protocols and frameworks, the request/response protocol used to obtain the assertions, and bindings of this protocol to various transfer protocols (for example, SOAP and HTTP).

3.2 Abbreviations and acronyms

This specification uses the following abbreviations:

API	Application Programming Interface
App	Application software
App ID	Application identifier
AS	Authorization Server
CDN	Content Delivery Network
CP	Content Provider
HTTP	Hyper-Text Transfer Protocol
ID	Identifier
IdP	Identity Provider
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
MVPD	Multichannel Video Programming Distributor
OAuth	Open Authorization Specification (IETF standard)

REST / RESTful	Representational State Transfer
RO	Resource Owner (OAuth Term)
RP	Relying Party (SAML Term)
RS	Resource Server (OAuth Term)
SAML	Security Assertion Markup Language
SP	Service Provider (SAML Term)
SSO	Single sign-on
TVE	TV Everywhere
UA	User Agent (typically a browser)
UI	User Interface
UN/PW	User Name and Password
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
XML	Extensible Markup Language

4 DEFINITIONS

For the sake of consistency, this section defines the entities and terms used in this document.

4.1 Entities

Table 1 - Entities in the Eco-System

CDN	Content Delivery Network – used by either MVPDs or Programmers to distribute their content.
Consortium	A group of MVPDs, Programmers, or a combination. A consortium may be created to consolidate some functionality common across the participants.
MVPD	Multi-channel Video Programming Distributor – A fee based distributor of video content, frequently from different Programmer sources.
Programmer	A developer and provider of specific video content.
Subscriber	An individual who is a paid subscriber to video content.
Third-Party	As used in this document, third- party refers to an application developer that is neither an MVPD nor a Programmer.
User	Synonym for Subscriber.

4.2 Roles

Table 2 - Roles and their Players

Role	Definition	Can be played by
Application Developer	Applications are developed by an entity called Application Developer.	3rd party, MVPD, Programmer
Application Admin	Applications need to be regulated – definitely, the services exposed by MVPDs and Programmers are expected to be consumed by pre-registered entities, possibly with business agreements in place. The 'Application Admin' would be that regulator. The responsibilities include: • Issuing Identities (and credentials) to Application Developers • Managing the life-cycle of application identities • Managing the privileges/authorizations provided to each application • Run-time: Validating application's credentials and issuing tokens.	MVPD, Consortium
CP	Content Provider – an entity that actually streams video content to the subscriber's device.	MVPD, Programmer, CDN
IdP	Identity Provider – an entity that authenticates users/subscribers. This entity has subscriber's account information and credentials. It can validate/authenticate the user. This entity manages user's account life-cycle.	MVPD
SP	Service Provider – an entity that exposes certain services that can be used by the application. And MVPD or a Programmer could be a Service Provider.	MVPD, Programmer

5 OVERVIEW

The following figure depicts pictorially the flow outlined in Section 1.3.

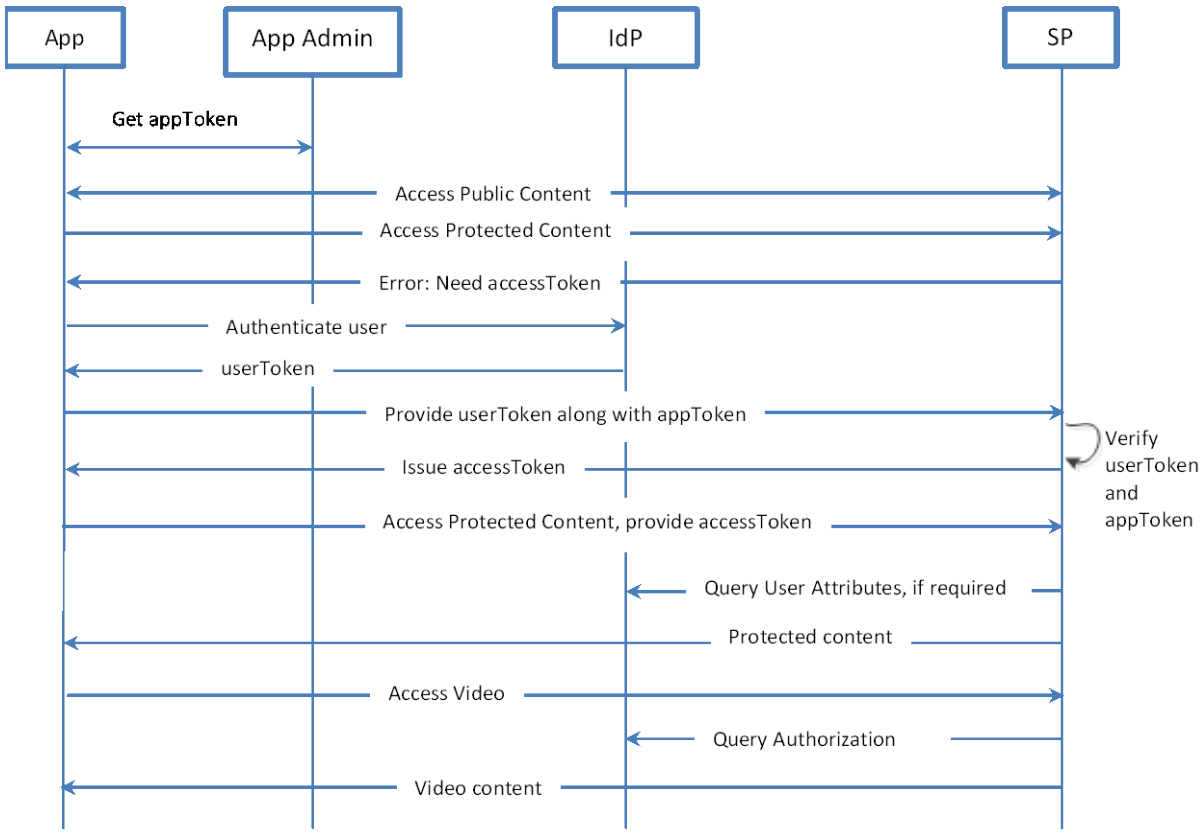


Figure 2 - High-level Flow of Messages

The following are comments regarding Figure 2.

1. App knows the App Admin to which it needs to authenticate.
2. App authenticates itself (with the embedded 'secret', previously issued by App Admin as a result of successful registration, or some other technique prescribed by App Admin).
3. App Admin issues an 'appToken' to the App that proves that the App is a legitimate app. The appToken may contain claims belonging to the app, including the privileges (see below).
4. SP may provide some content as public content – no tokens needed to access that type of content.
5. When App tries to access a protected content, SP will respond with an error that requires an accessToken.
6. App follows procedures outlined in this document to obtain an access token.
7. App presents that accessToken on all further requests.
8. SP may query the IdP for any subscriber attributes.
9. SP will grant access to the protected content.
10. For actual content access (playing a video), SP will perform an authorization with the IdP. On approval, SP may either stream the content itself, or redirect to a CP – giving a contentToken that proves to the CP that this is an authorized request.

Note on End-Points: App uses end points provided by different entities for various purposes – accessing content, authenticating the user, etc. This document does not define how the App obtains such end points. It is up to the developer to get such information either statically (at the time of packaging the application), or discover dynamically.

Note on trust relationships: The architecture proposed here relies on the ability of entities consuming the tokens to be able to validate their authenticity (through the signature included in the token). However, how the entities obtain the trust relationship for such signature validation is not covered in this document. It is up to the deployments to identify which entities they would trust and obtain the credentials from those entities.

5.1 Requirements

Table 3 lists technical and functional requirements considered when defining the architecture.

Table 3 - List of High-level requirements

Req #	High Level Requirement	Description
App.1	App Authentication	All applications must be authenticated
App.2	App Authorization	Applications must be associated with privileges, and the service providers must have a way to verify the privileges belonging to an application.
User.1	User Authentication	Users must be authenticated using MVPD provided IdP.
User.2	No MVPD Application necessary	User Authentication must not depend on the availability of an MVPD provided application on the device.
User.3	Username/password based authentication	Applications must support username/password based authentication to verify user's credentials.
User.4	SSO across applications	Solution should support Single sign-on across multiple applications that the user may be using on the same device.
User.5	Logout	Solution must support user logout.
AuthZ.1	Content access authorization	CPs/SPs must authorize the specific content (by the MVPD) before it is played for the user.
Gen.1	No browser	Solution must support at least one way for applications running on devices that do not have browsers.
Gen.2	Standards	Use well-known standards where applicable.
Gen.3	Existing infrastructure	Preference should be given to utilize existing infrastructure as much as possible.

5.2 Assumptions

The following assumptions are made in this specification.

Table 4 - List of Assumptions

#	Description
1	Applications can make HTTPS connections, particularly with server certificate validation.
2	Applications have the capability to hold (embedded) credentials securely. Credentials could be private keys, shared secrets, or other.
3	Applications are capable of performing crypto operations like encryption/decryption/signatures, with minimum strengths identified in this document.
4	Applications can process JSON (though not necessarily XML).
5	Applications have access to (secure) local storage that allows them to store some (dynamic) data.
6	If provided by the environment, applications can display browser within the context of the application (like WebView on iOS).
7	Applications understand the (pre-published) privileges.

5.3 Solution Discussion

In Section 6, the architecture is discussed in detail. In this section, how the architecture came about is covered.

The flow depicted in Figure 2 requires consideration of the following technical requirements.

1. Concept of a token, and different types of tokens required.
2. Protocols:
    a. For the server to ask for certain types of tokens when a resource request is received, and corresponding protocol for the client to supply those tokens
    b. To obtain a specific type of token

5.3.1 Tokens

5.3.1.1 Token Definition

This architecture is based on the concept of a token. A token is defined as a set of claims about a specific entity, signed by an issuing entity, the signature of which can be verified by a relying party (who has pre-established trust with the issuing entity). It is assumed (by the relying party) that the issuing entity has performed adequate authentication of the specific entity before issuing the token to it.

Token = {Set of Claims} + {Signature of the issuer}

The set of claims is further defined to include the following information

Set of Claims = Issuer, Issue Timestamp, Expiry Timestamp, Subject, Audience (optional)

These fields are defined in Table 5 below.

Table 5 - Fields in a Token

Field	Definition
Issuer (Mandatory)	The universally unique Identifier for the entity that issues the token. This specification does not define the format for this identifier, but it is assumed that all participants in the eco-system have a way to communicate their respective identities to other (and related trust mechanisms) through some mechanism. Such mechanisms are outside of the scope of this document.
Issue Timestamp (Mandatory)	The time at which the token is issued. This is required to be UTC time. The actual value is the number of milliseconds from the epoch.
Expiry timestamp (Mandatory)	The time at which the token expires. This is required to be UTC time. The actual value is the number of milliseconds from the epoch.
Subject (Mandatory)	The universally unique identifier of the entity to which this token is issued. Again, the format used for achieving universal uniqueness is not defined in this document.
Audience (Optional)	The universal unique identifier of the entity to whom this token is addressed. This can be used to specifically constrain the token usage.

A token may contain other fields. Section 6 identifies those fields based on context.

5.3.1.2 Token Types

The architecture presented in Section 6 uses the types of tokens listed in Table 6.

Table 6 - Types of Tokens used in the Solution

Token Name	Definition	Issued By	Intended Audience
appToken	A token issued to an application by the Application Administrator, mostly as a result of a successful authentication of the application credentials. This token may also include privileges (in terms of which class of web services it can access). The format of this token is standardized in this specification.	App Admin	SP, IdP
userToken	A token issued as a result of successful user/subscriber authentication. This token is issued by the IdP, and could be delivered to the App or the SP. The format of this token is standardized in this specification.	IdP	SP, App
accessToken	An OAuth access token. See [ID-OAuth]. The format of this token is NOT standardized in this specification. This token is opaque to the client, and meaningful only to the SP.	SP [AS]*	SP [WS]*
contentToken	A token issued by the SP to verify that a request for a particular content is an authorized request. This token is issued by an SP and targeted for a CP. The format of this token is standardized in this specification.	SP	CP

NOTE: * See Table 7 for further discussion.

Notice that only tokens that are exchanged across domains are standardized in this specification. Tokens used within a single domain are opaque and not standardized in this specification.

5.3.1.3 Token Formats

In the industry, there are two token formats that are being widely adopted – one based on SAML and another based on JSON. This document supports both SAML Assertions based on [SAML Core] and JSON Web Tokens based on [ID-JWT].

Only the accessToken is an opaque token (format not defined in this document). Formats for all other tokens are defined in this document.

5.3.1.4 Token Structure

The issuer of the token denotes the format of the token as well. Thus, a response containing the token is a JSON structure consisting of two mandatory fields:

tokenFormat

tokenContent

Note that the *tokenContent* field contains the actual token, and the *tokenFormat* specifies the format of the content in *tokenContent* field. Valid values for *tokenFormat* are *JWT* and *SAML*.

5.3.2 Protocols

Broadly speaking, protocols are needed to ask or trigger a user/app authentication, and provide for the transport of different tokens on resource requests.

OAuth 2.0 ([ID-OAuth]) specification has extensive coverage of use-cases in the Apps' world. It prescribes protocols to initiate user authentication, and get authorizations for accessing user's resources on a resource server. It defines the bindings to use and parameters to communicate the tokens. It would be ideal if OAuth could be adopted as is.

However, there are fundamental differences between OAuth entities and entities in TVEverywhere (TVE) (see Section 4). For one, in OAuth, it is the user who 'allows' access to 'their' content to an application. In TVE, it is the MVPD (through the IdP role) that 'allows' access to a resource for a specific user (at a specific time). Also, OAuth explicitly does not define the token formats. In TVE, the notion of user's identity (however opaque) is important to SPs so that they can perform personalization – to name one feature. Also, OAuth does not define how to perform application (client) authentication, the standard does not include bindings to include application tokens (though sister standards do).

OAuth, though, has one important advantage – defining interfaces that are simpler, piggy-back on HTTP protocol, and do not explicitly require XML processing on the App side.

Given the advantages, and disadvantages of OAuth, this specification 'adapts' OAuth – allowing us to be 'compatible' and still meet our needs. The solution provided in this document melds SAML-based user authentication, back-channel content authorization between the SP and IdP, and OAuth for App communication with IdP/SP/CP.

Table 7 defines the mapping between OAuth roles and TVE roles.

Table 7 - Mapping of OAuth Roles to TVE Roles

OAuth Role	Maps to	TVE Role
Client	→	App
User Agent [UA]	→	Browser (either stand-alone browser or through WebView kind of browser)
Resource Owner [RO]	→	Subscriber Note that this mapping is ONLY to perform user authentication. Resource Server/SP still perform a back-channel authorization with MVPD for every content access. However, mapping OAuth RO to TVE Subscriber allows us to stay true to OAuth user authentication flow.
Authorization Server [AS]	→	An implementation at Service Provider (Again, this mapping is made for protocol purposes, not to mean that Service Provider acts like the Authorization Server for the actual content access. MVPD still provides the content authorization through back channel.) The objective here is to let the SP create the Access Tokens. This makes sense as the SP is the consumer of Access Tokens, and having the SP create the Access Tokens enables flexibility on how the SP defines/creates/manages these Access Tokens.
Resource Server [RS]	→	Web Services Server at Service Provider / Content Provider
NOTE: In addition, 'scope' from OAuth is used specifically for privileges belonging to the App.		

6 ARCHITECTURE

This section maps the high-level flow to chosen technologies to meet the requirements identified earlier in this document. It also identifies which of the messages from the high-level flow map to OAuth flows, and semantics for the request and response parameters.

The approach taken in this document is 'derived' from the OAuth [ID-OAuth] specification. Figure 3 outlines the high-level flow mapped to OAuth.

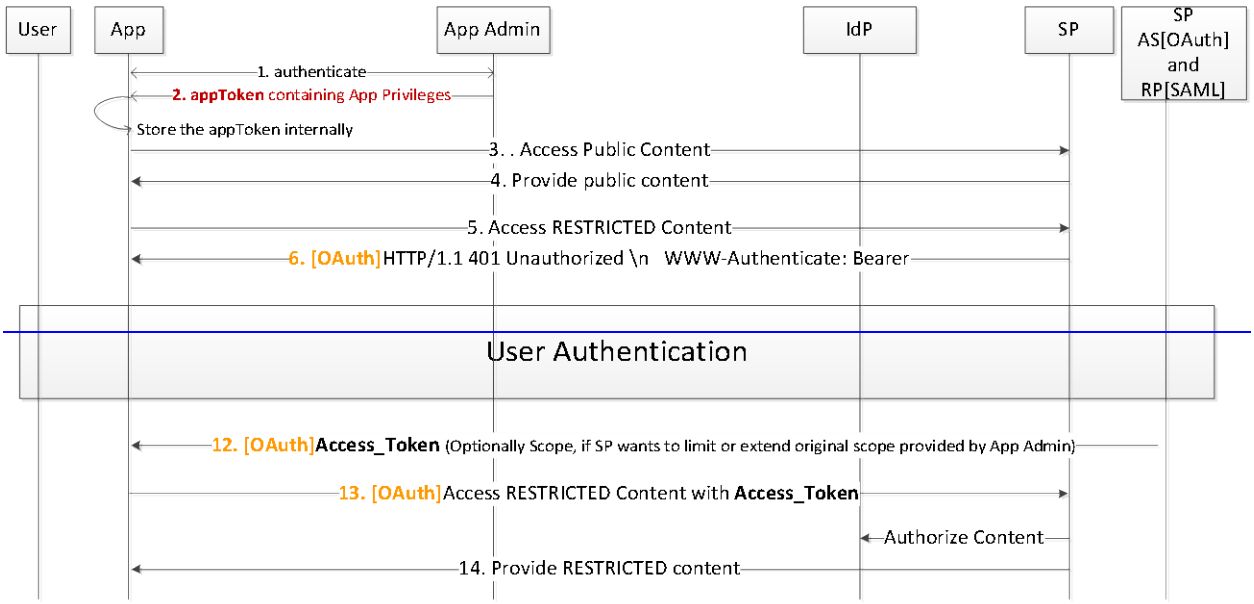


Figure 3 - High-level Flow Mapped to OAuth

Steps where the OAuth protocol is specifically used are tagged with the prefix of [ID-OAuth] in the figure above.

How an OAuth Access Token is requested is not shown in Figure 3. How that happens depends on how the user authentication happens. For user authentication, two forms are supported:

1. (Existing) SAML Protocol – requires browser
2. Custom API (to validate a given username/password) – does not require browser

Flows for both SAML-based and API-based User Authentication are shown in the following two figures. SAML specific flow elements in Figure 4 are prefixed with [SAML], and API specific flow elements in Figure 5 are prefixed with [Custom API].

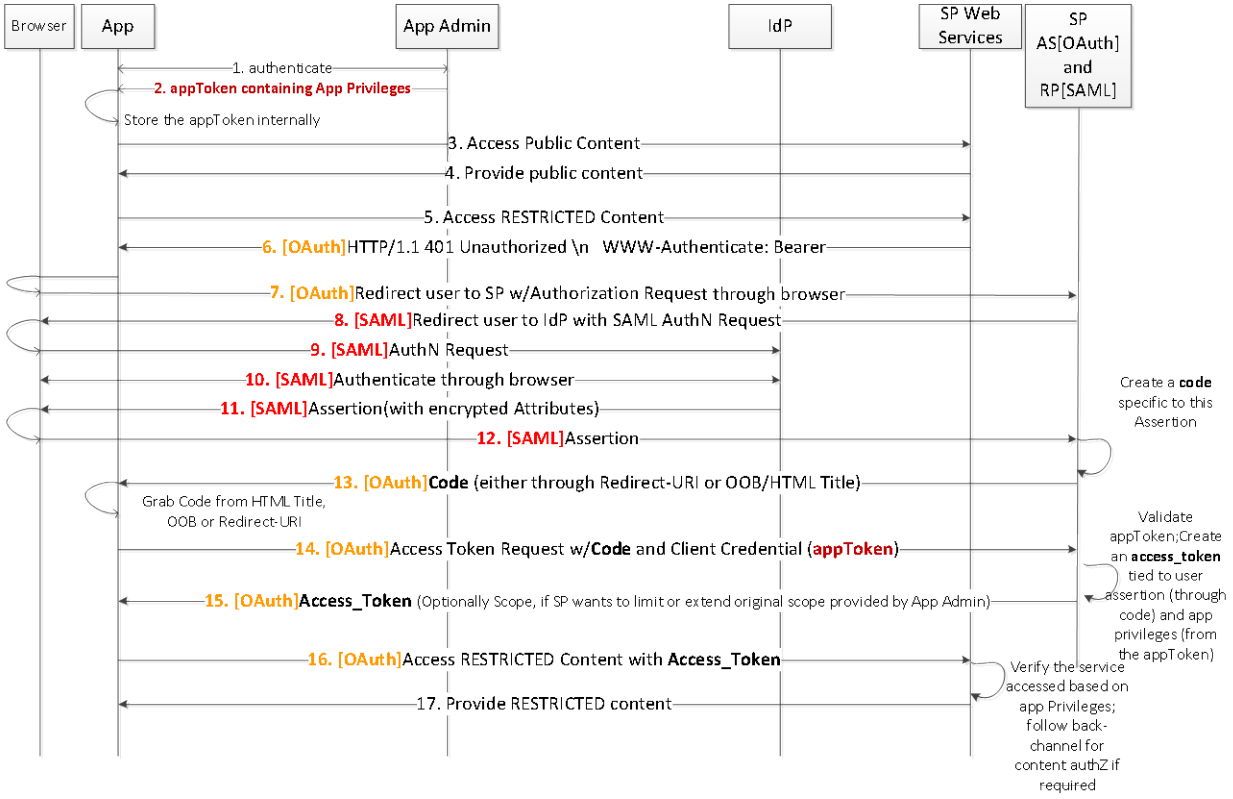
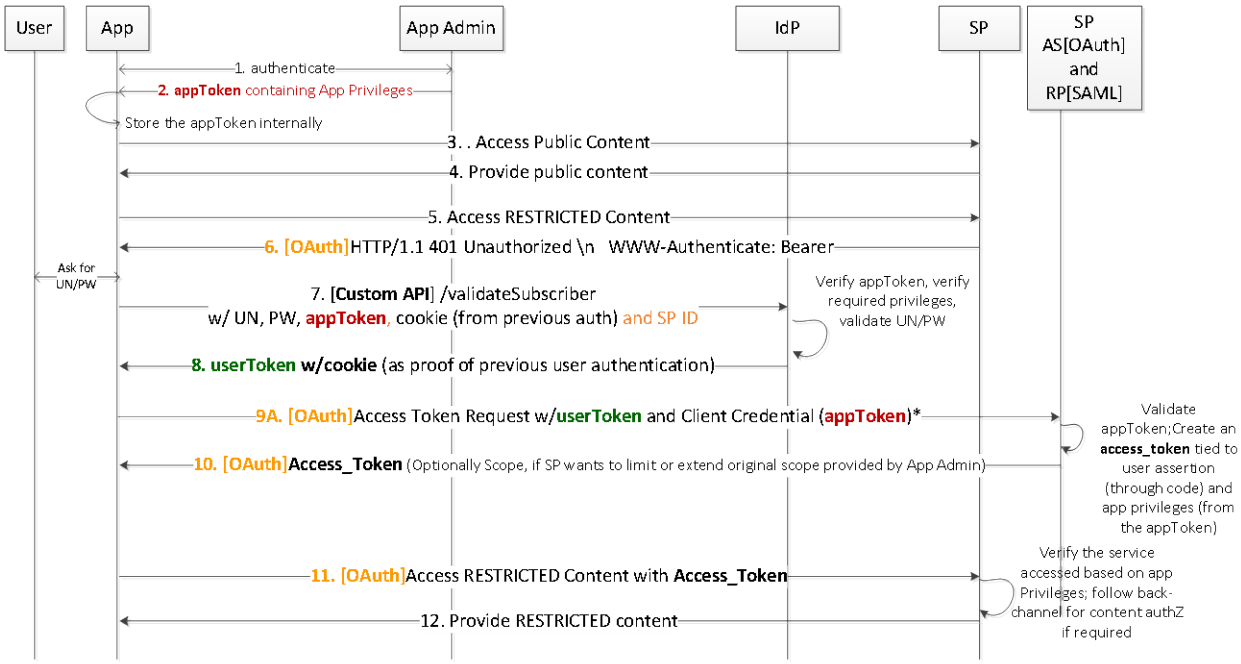


Figure 4 - SAML (Browser)-Based User Authentication



NOTE: *See the Using Assertions as Authorization Grants section of [ID-OAuth Assertions].

Figure 5 - API-based User Authentication

Steps in the above flow diagrams are discussed below.

1. App is approved by a specific App Admin – each App knows how to contact their App Admin. Application also knows how to authenticate itself to App Admin. These details are not covered in this specification.
2. App Admin issues an appToken upon successful authentication by the App. The appToken is standardized in this specification. The appToken is signed by the App Admin according to this specification. Any one that has a trust relationship with the App Admin will be able to verify this token. However, SPs are especially called out as requiring this trust relationship. How this trust is established is not covered in this specification.
3. AppTokens may include privileges belonging to the App.
4. SPs may expose services that do not require any App or User authentication.
5. SPs are gateway to the content – thus they need to ensure that the App to which they are providing the content is an authorized App.
    a. Beyond App verification, SPs need to continue user-based Authorization checks at the time of content access.
    b. Thus, SPs need two types of tokens to deliver any content – appToken and userToken.
6. SPs may need user information for:
    a. Back-channel authorization with the Authorization Providers
    b. Personalization, or any other value added services they want to provide
    c. Presentation of content based on user attributes
7. This specification uses OLCA Authorization protocols to perform back-channel authorization checks and/or attribute queries.
8. OAuth protocols are used between the App and SP. In this specification, SPs act like both an Authorization Server and Resource Server. A typical OAuth based flow follows:
    a. App accesses a service that is protected.
    b. SP responds with an OAuth error message (see the WWW-Authenticate Response Header Field section of [ID-BearerTokens]).
    c. Based on the Authorization Code section of [ID-OAuth], App 'redirects' the user to (SP's) Authorization Server with an OAuth Authorization Request. App may use either an external browser or WebView kind of browser.
    d. SP needs to authenticate the user – it creates a SAML AuthNRequest and redirects the user agent to IdP. IdP authenticates the user and redirects the user back to SP's Authorization Server with a SAML Assertion. This particular sequence is completely based on the OLCA Authentication specification [AUTH1.0].
    e. When SP's Authorization Server receives the SAML Assertion, it extracts the user information, creates an Authorization code and redirects the user back to App's redirection URL with that code.
    f. App obtains the code and contacts SP's Authorization Server with the code and appToken for an accessToken.
    g. SP's Authorization Server validates the signature on the appToken, extracts the privileges (scope) and creates an accessToken and sends it back to the App. Authorization Server may change the scope for this access token (from what is specified in the appToken – either increase or decrease). If so, the response to the App also contains the new scope.
9. If a custom API is desired instead of the above-described SAML flow for user authentication, then the way an App gets the Access Token is based on OAuth Assertion Profile [ID-OAuth Assertions].
    a. App obtains the username and password from the user directly and calls a custom API provided by the IdP.

b. This specification defines the custom API.

c. App sends its appToken on the custom API, along with the SP's entity ID.

d. IdP validates the appToken, and verifies the UN/PW combination. If valid, IdP creates a userToken and sends it back to the App. The userToken could be SAML Assertions based or JWT based.

e. App sends that userToken to SP's OAuth Authorization server's token end point (along with appToken) – based on the Using Assertions as Authorization Grants section of [ID-OAuth Assertions].

f. SP's OAuth Authorization Server validates the userToken AND appToken, and responds with the Access Token. As stated earlier, SP may enhance or diminish the privileges it wants to give the App, in which case it also includes a scope parameter in the Access Token response.

10. Apps will use the Access Token from then on for any requests they make to that SP. Since SPs have an associated appToken and userToken for each Access Token, they can verify if an App is allowed to invoke the service in context, and/or if the user from the userToken has access to the data/content being requested.

    a. The Accessing Protected Resources section of [ID-OAuth] discusses how to send the Access Token on service requests. This specification uses the Bearer tokens, specifically using the Authorization header, as detailed in the Authorization Request Header Field section of [ID-BearerTokens].

6.1 App Authentication

This section covers the App Authentication flow segment.

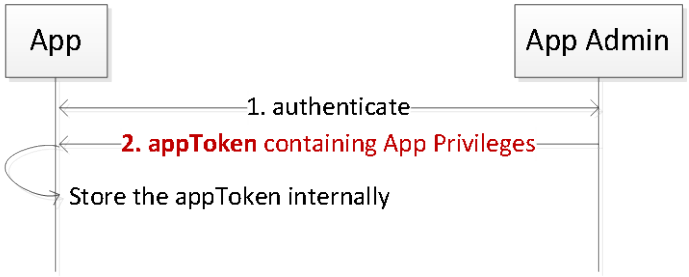


Figure 6 - App Authentication Flow

Application authentication happens between the App and the App Admin. There are two distinct steps to this.

1. Application uses a protocol specified by the App Admin to prove its credentials.
2. App Admin creates an appToken and delivers that to the App

This document does not define the first step. Also, the URL (or any form of address) to contact App Admin is not specified; it is up to the App Admins. In the second step, while this document does not define the specific protocol by which the App Admin delivers the appToken to the App either, it does define the format for the appToken.

6.1.1 Format of AppToken

An appToken is a JSON-style object. It has the following schema.

```
/* [M} stands for a mandatory field, [O] stands for an optional field */
{
[M] "tokenFormat" : "Format of the attached token – either SAML or JWT",
[M] "tokenContent" :  "Actual token – either in the JWT format or in SAML format",
[O] "expires" : "in Milliseconds from epoch", /*Note this is for the sake of the App */
[O] "scope" : ["Privilege-1", "Privilege-2"] /* Note this is for the sake of the app, same information
may be repeated inside of actual token. */
}
```

6.1.1.1 JWT Format

In this specification, the JWT Compact Serialization in [ID-JWT] is used. Per that specification, a JWT token has the following segments:

1. Header segment,
2. Payload segment, and
3. Crypto segment

The segments are separated by a period ".". Header and crypto segments MUST be per [ID-JWT]. The payload section is defined in this document.

The JWT payload has the following schema.

```
{
  [M] "iss": "App Admin Entity ID",
  [M] "urn:cablelabs:app:1.0:token:issueTime" : "Issued time in milliseconds",
  [M] "exp" : "Expires time in milliseconds",
  [M] "urn:cablelabs:app:1.0:token:type" : "urn:cablelabs:app:1.0:token-type:appToken",
  [M] "urn:cablelabs:app:1.0:token:subject" : "App ID, that the App also knows",
  [O] "urn:cablelabs:app:1.0::privileges" : ["Privilege-1", "Privilege-2"]
}
```

The App Admin MUST include "iss", "urn:cablelabs:app:1.0:token:issueTime", "exp", "urn:cablelabs:app:1.0:token:type", and "urn:cablelabs:app:1.0:token:subject" attributes in the appToken payload. App Admin MAY include "urn:cablelabs:app:1.0::privileges" in the appToken payload.

A description of the fields is given in the table below.

Table 8 - Field in AppToken Payload

Field	Description
"iss"	Entity ID of the issuing App Admin. This entity ID is a globally unique identifier. "iss" MUST be in a valid URI format. (Note that this specification does not define how uniqueness is achieved, it is up to the App Admins.) It is important to notice that this ID is going to be treated as a key into the trust relationship that the receiver of this token has about this entity. Thus, this ID must relate to a trust relationship.
"urn:cablelabs:app:1.0:token:issueTime"	The time this token is created/issued, in UTC time zone. The value is the number of milliseconds from January 1, 1970 00:00:00 UTC.
"exp"	The time this token expires, in UTC time zone. The value is the number of milliseconds from January 1, 1970 00:00:00 UTC.
"urn:cablelabs:app:1.0:token:type"	Indicates the type of token. For appTokens, the value MUST be "urn:cablelabs:app:1.0:token-type:appToken".
"urn:cablelabs:app:1.0:token:subject"	Application ID. This is the unique identifier given to this application by this App Admin. This ID uniquely identifies the application to which this token is issued. The entity receiving this token may or may not know this ID. It is beyond the scope of this document how the App Admin may communicate the Application ID to other entities in the eco-system.
"urn:cablelabs:app:1.0::privileges"	A (JSON) array of privileges belonging to this application (granted by the App Admin). See Section 6.1.2 below for the semantics of App Privileges.

6.1.1.2 SAML Format

The tokenContent is formed of the following syntax (per the "Deflate Encoding section of [SAML Bindings]):

SAMLResponse=value&SigAlg=value&Signature=value.

The whole string MUST be URL-encoded, as ampersands and equal signs here will collide with ampersands when this token is being transmitted as part of a URL.

App Admin MUST include SAMLResponse, SigAlg, and Signature fields.

App Admin MUST create the values for SAMLResponse, SigAlg, and Signature fields as per Section 3.4.4.1 of [SAML Bindings].

Subject NameID in the SAML Assertion MUST be the Application ID.

Issuer of the SAML Assertion MUST be the entity ID of the App Admin.

The SAML Assertion MUST have an AuthenticationStatement.

The SAML assertion SHOULD have an AttributeStatement consisting of (at least) the following attribute

urn:cablelabs:app:1.0::privileges

That attribute can have multiple values – one for each privilege that the application has.

6.1.2 Application Privileges

The goal of the App Admin is to manage App's credentials and tokens. Additionally, it may also administer privileges provided to Apps.

The semantics of such privileges need to be pre-established between the App Admin and the SP (and IdP in some use cases). This specification recommends the following:

1. Privileges be pre-defined (privileges could be "schedule a VOD recording", "change a channel", "buy VOD content", etc.).
2. Give a name to each privilege.
3. Publish those names (and semantics) to the SPs/IdPs.
4. Access is binary – either an application has a particular privilege, or not.
5. Privileges be included in the appToken. This way, when the SP receives the appToken, it can see what it is allowed to do.

This specification does not define the privileges themselves; they are determined by the App Admins / SPs.

6.1.3 Storing the AppToken

Applications are expected to store the appToken internally. They have access to the "expires" and privilege parameters in the appToken JSON structure. Apps may use these values to intelligently decide when to get a new token and whether they can perform certain actions or not. This document, of course, defines the error messages the App can expect if it uses an expired token, or invokes a service to which it is not privileged.

6.1.4 Error Conditions

Error conditions for the App Authentication flow are not specified in this document. It is up to the App Developer and App Admin.

6.2 Unauthenticated Access

This section covers the unauthenticated access flow segment as shown below in Figure 7.

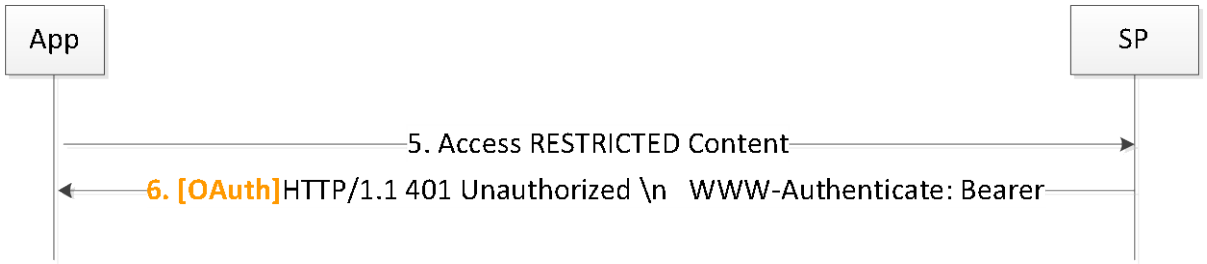


Figure 7 - Unauthenticated Access

SPs MUST respond with a 401 code (as described in the WWW-Authenticate Response Header Field section of [ID-BearerTokens] when they receive request for a service that requires an Access Token. The actual response should look like the following:

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Bearer realm="<<Realm Name>>"

Note: Realm is not semantically defined in this specification.

See Table 12 - Error Codes and Expected Behavior for specific error codes to use for various conditions.

When App receives a 401 asking for a bearer token (see Section 6), it should trigger a process to obtain an accessToken. A typical [ID-OAuth] flow is used to obtain the accessToken. This includes performing user authentication, obtaining a code as a result of successful user authentication, and presenting the code along with client credentials (appToken) to obtain the accessToken.

The following sections cover these steps.

6.3 SAML-based User Authentication

This section covers the SAML-based User Authentication flow segment.

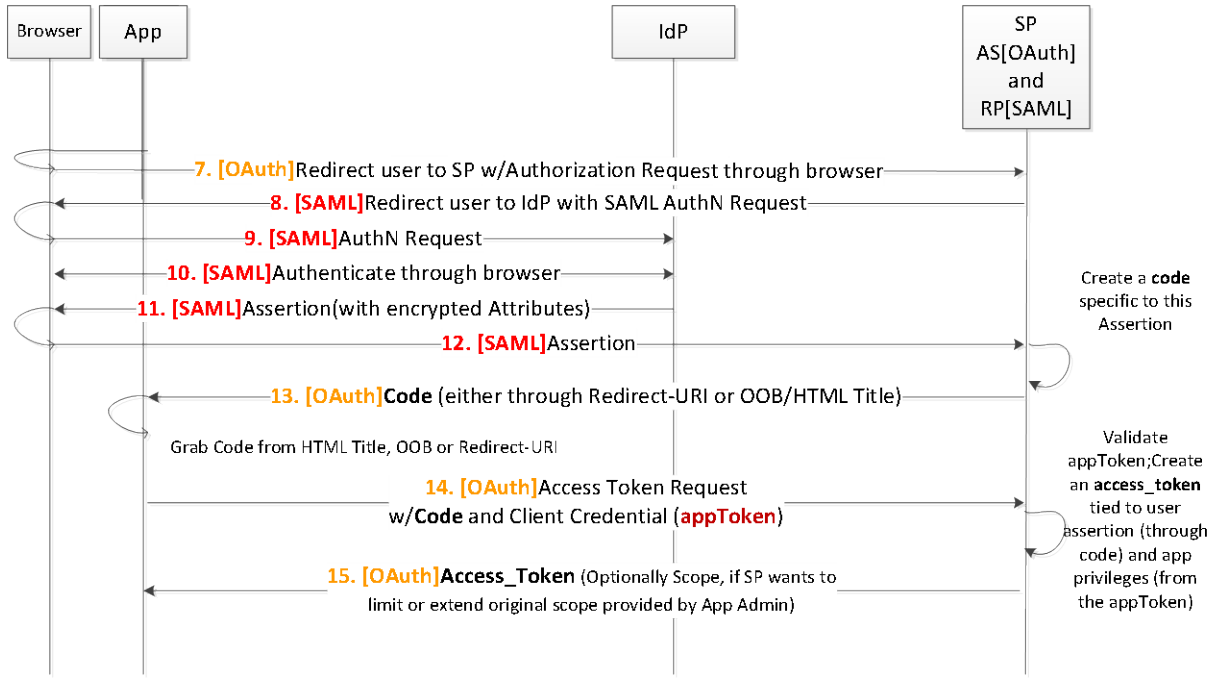


Figure 8 - SAML-based User Authentication Flow

6.3.1 Authorization Request

This section covers Step 7 of Figure 8. The App creates an OAuth Authorization Request and redirects the user through a browser (internal or external) to the (SP's) Authorization Server. How the App obtains the Authorization Server's URL is outside the scope of this document.

The App MUST adhere to the "Authorization Request" section in [ID-OAuth] for the authorization request. In addition, the following rules apply:

1. client_id MUST be App ID (must match "urn:cablelabs:app:1.0:token:subject" in appToken)
2. scope SHOULD not be included. Authorization Servers can ignore scope if it is included.
3. redirect_uri MUST be included.

6.3.2 SAML-based User Authentication

This section covers Steps 8 through 12 of Figure 8. Upon receiving the Authorization Request, the SP's AS MUST generate a SAML Authentication Request and redirect the user agent (browser) to IdP. The SP may ask the user to select their IdP before redirecting the user. (An SP may also employ techniques to auto-detect the IdP – such techniques are outside the scope of this document.)

The SP MUST generate SAML AuthNRequest and redirection procedures according to [AUTH1.0].

The IdP will perform user authentication, generate a SAML Assertion and redirect the user's browser back to SP. The IdP MUST meet the requirements in [AUTH1.0] for the generated SAML Assertion and redirection to SP. Additionally, it is highly recommended that any attributes included in the Assertion be encrypted (in accordance with [AUTH1.0]), to prevent the App from reading the attributes from the Assertion.

6.3.3 Authorization Response

This section covers Step 13 of Figure 8. After receiving an assertion for successful authentication, the Authorization Server extracts the necessary user information from the assertion and creates a code keyed to that information. It redirects the user agent back to App with the code. The AS MUST adhere to the Authorization Response subsection

of the Authorization Code section in [ID-OAuth]. Note that the URL to which the AS redirects with the code is the redirect_uri given in the Authorization Request [Step 7]. It is up to the App implementation to be able to receive or intercept this redirect.

6.3.3.1 Error Conditions

All implementations MUST adhere to the Error Response section of [ID-OAuth]. In addition to the error codes identified in that section, the following error codes are permitted.

Table 9 - Possible Error Codes Sent in Authorization Response

Error Code	Meaning
invalid_user	User authentication failed.
user_not_authorized	User is not authorized for this SP's content.

The AS MAY include error_description and error_uri in the response. If they are included, the App MUST show them to the user.

6.3.4 Access Token Request

This section covers Step 14 of Figure 8. With the code received in the Authorization Response [Step 13], App will contact SP's AS for an accessToken. This will be an OAuth Access Token request.

- AppToken MUST be included in parameters called client_assertion_type and client_assertion.

From the appToken (see Section 6.1.1), the value of the "tokenFormat" MUST be sent in a parameter called client_assertion_type, and the value of the "tokenContent" MUST be sent in a parameter called client_assertion. See the Transportation Assertions and Client Authentication sections in [ID-OAuth Assertions].

6.3.5 Access Token Response

This section covers Step 15 of Figure 8. SP's AS will validate the appToken signature and verify that the code in the request was given to this App. SP's AS will then create an accessToken, tied to data about the user from the SAML Assertion, and data about the App from the appToken. It is essential that the AS does this binding between the accessToken and data from Assertion and appToken, as when the accessToken is used for subsequent resource access, SP can determine if that request is authorized or not.

The AS MUST adhere to the Access Token Response and Successful Response sections of [ID-OAuth] for the access token response. In addition, the following rule applies.

- "scope" parameter MAY be present in the response JSON structure.

If "scope" parameter is present, App SHOULD use that as guidance for what privileges it has at this particular SP, rather than the global privilege set it obtained from the App Admin through appToken.

6.3.5.1 Error Responses

All error responses MUST be according to the Error Response section of [ID-OAuth].

6.4 API-based User Authentication

This section covers the API-based User Authentication flow segment.

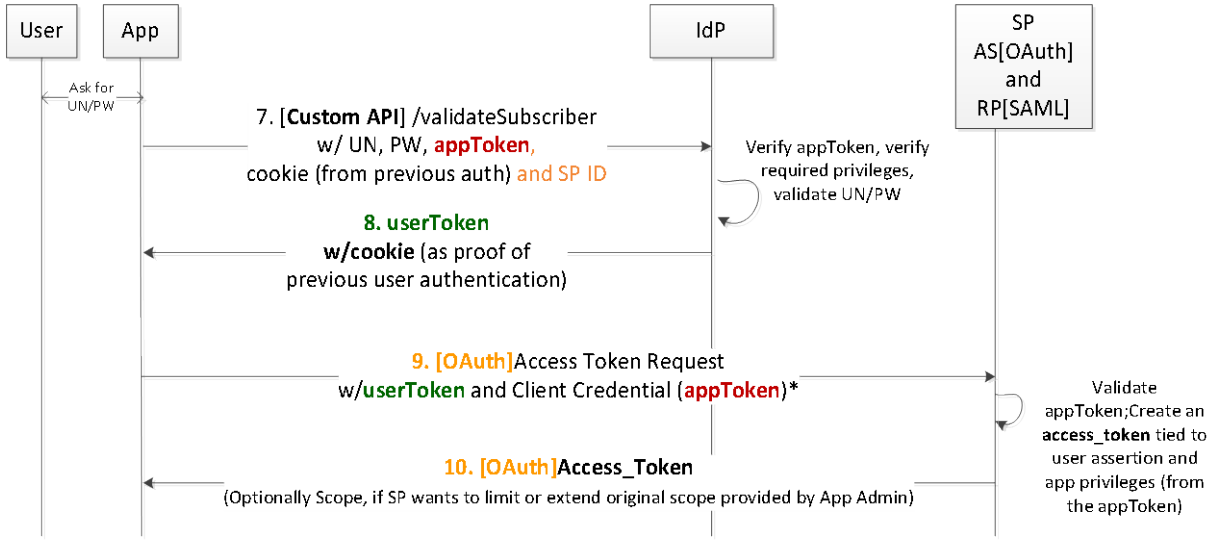


Figure 9 - API-based User Authentication flow

*See the Using Assertions as Authorization Grants section of [ID-OAuth Assertions].

6.4.1 Subscriber Validation API

This section covers Step 7 of Figure 9. The App obtains username and password of the subscriber through its own user interface (UI). IdP provides a web service (over secure channel) that allows the App to validate the entered username and password. It is expected that the IdP would provide this validation service only to trusted applications – thus, the App needs to provide its appToken in the request.

6.4.1.1 Request

Request API has the following signature.

URI – custom

Content-type: application/x-www-form-urlencoded;charset=UTF-8

Allowed HTTP POST Parameters: username, password, client_assertion_type, client_assertion, spEntityID

Cookies from previous authentication

The HTTP parameters are described in Table 10 below.

Table 10 - Parameters in a Subscriber Validation API

Parameter	Requirement Type	Description
username	Mandatory	Username entered by the user
password	Mandatory	Password entered by the user
client_assertion_type	Mandatory	Value of parameter "tokenFormat" from the appToken

Parameter	Requirement Type	Description
client_assertion	Mandatory	Value of parameter "tokenContent" from the appToken
spEntityID	Optional	Target SP Entity ID; IdPs may entertain a request without a specific target SP.

6.4.1.2 Response

IdP validates the request and may find errors in the request, either relating to the request itself or with the given combination of username and password. If no errors are present, IdP creates a 'userToken' and includes it in the response. The response has the following JSON signature.

{
[M] "tokenFormat": "Valid values are JWT, and SAML",
[M] "tokenContent" : "Actual token, either in SAML or JWT format"
}

6.4.1.2.1 JWT Format

See "JWT Format" under section "App Authentication". The JWT payload schema is given below. IdP MUST create the token in the following format.

{
[M] "iss": "IdP Entity ID",
[M] "urn:cablelabs:app:1.0:token:issueTime" : "Issued time in milliseconds from January 1, 19700:00hrs in UTC",
[M] "exp" : "Expires time in milliseconds from January 1, 1970 00:00hrs in UTC",
[M] "urn:cablelabs:app:1.0:token:type" : "urn:cablelabs:app:1.0:token-type:userToken",
[M] "urn:cablelabs:app:1.0:token:subject" : "Subject NameID that the IdP wants to send to this SP. This must be compatible with a Subject NameID the IdP would have sent in a SAML Assertion. SP MUST use this for back-channel authorization requests.",
[M] "urn:cablelabs:app:1.0:token:subject-format" : "Transient OR Persistent. See [SAML Core] for interpretation by IdP or SP.",
[O] "urn:cablelabs:olca:1.0:attribute:authz:channelID" : "Channel ID",
[O] "urn:cablelabs:olca:1.0:attribute:authz:maxMPAA" : "maximum MPAA rating for this user",
[O] "urn:cablelabs:olca:1.0:attribute:authz:maxVCHIP" : "maximum VCHIP rating for this user"
}

6.4.1.2.2 SAML Format

See "SAML Format" under section 6.1, App Authentication. The IdP MUST generate a SAML Assertion according to [AUTH1.0].

6.4.1.3 Error Response

An error response from IdP MUST follow the following JSON structure. The HTTP status code MUST be 401.

{
[M] "error" : "Error code",
[O] "userMessage" : "Optional error message, to be displayed to the user"
}

The Error Codes listed in Table 11 are allowed.

Table 11 - Possible Error Codes for Subscriber Validation Request

Error Code	Interpretation
spID_missing	SP Entity ID must be given
spID_invalid	IdP did not recognize the given SP Entity ID
unPW_invalid	Given username and password combination is invalid
client_invalid	Given appToken is not recognized by IdP
client_expired	Given appToken expired

6.4.2 Access Token Request

This section covers Step 9 of Figure 9. After getting a userToken from the IdP, the App then contacts (SP's) Authorization Server for Access Token.

6.4.2.1 Request

Implementations MUST adhere to section 4.2 of [ID-OAuth Assertions]. In addition, following rules apply:

1. The *grant_type* MUST be either JWT or SAML.
2. The *assertion* MUST be the value of tokenContent.
3. Request MUST contain *client_assertion* and *client_assertion_type* (per the Using Assertions as Authorization Grants section of [ID-OAuth Assertions] and as described in Access Token Request, [Step 14], in Section 6.3.

6.4.2.2 Response

See Access Token Response, [Step 15] in Section 6.3.

6.5 Using Access Token

Apps use the accessToken for any requests they make to that SP. Since SPs have an associated appToken and userToken for each accessToken, they can verify if that App is allowed to invoke the service in context, and/or if the user from the userToken has access to the data/content being requested.

6.5.1 Request

The Accessing Protected Resources section in [ID-OAuth] discusses how to send the Access Token on service requests. This specification uses the Bearer tokens, specifically using the Authorization header, as detailed in the Authorization Request Header Field section of [ID-BearerTokens]. Implementations MUST adhere to the Authorization Request Header Field and the WWW-Authenticate Response Header Field sections of [ID-BearerTokens]. A simple example is shown below.

GET /resource HTTP/1.1
Host: sp.programmer.com
Authorization: Bearer <<AccessToken>>

6.5.2 Response

If the given accessToken is valid, SP will respond with the functional data for the service requested. Based on the access code, SPs are expected to be able to retrieve the associated user and App tokens (or the data in those tokens). SPs are expected to perform authorization checks for the requested content with the MVPDs, as well as make sure

that the privileges belonging to the App allow this particular request. If there are any errors found, SP MUST respond with a HTTP 401 error and MUST include an error parameter that identifies what is wrong with the request (see Error Conditions below). SP MAY also include a user_message. Apps MUST show the contents of user_message to users. A simple example is shown below.

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="<<Realm Name>>"
error="invalid_token",
user_message="Optional description"

This specification does not define the semantics of the realm parameter on the WWW-Authenticate header.

6.5.2.1 Error Conditions

The following error conditions are recognized:

1. Unrecognized token: error = "invalid_token"
2. Expired access token: error = "expired_token"
3. Expired userToken: error = "expired_user_token"
4. Expired appToken: error ="expired_app_token"
5. App not privileged for this web services call: error = "insufficient_scope"
6. User not authorized: error = "not_authorized". If any user_message is present in the response, then App MUST show it to the user.

Table 12 summarizes the expected App behavior upon receiving the above error_codes.

Table 12 - Error Codes and Expected Behavior

Error Code	Meaning	Expected Behavior / Notes
invalid_token	SP could not understand or parse the token.	This may require further investigation on the developer and SP sides.
expired_token	Access token expired.	App must obtain a new accessToken. If SP had given a refreshToken earlier, it may use that to obtain a new accessToken. If not, for API based user authentication, App may try an accessToken request with previous userToken and appToken. For SAML based user authentication, App should start with a new Authorization Request (that triggers user authentication).
expired_user_token	User must be re-authenticated.	App must initiate a new user authentication flow.
expired_app_token	App token expired.	App must contact its App Admin to get a new appToken. After that, App needs to get a new accessToken based on this appToken. For API-based user authentication, it may use an existing userToken to make an accessToken request. For SAML-based user authentication, App must start with a new Authorization request (that triggers user authentication).

Error Code	Meaning	Expected Behavior / Notes
insufficient_scope	App does not have privileges associated with the service requested by it.	Typically, this results in an error message to the user, indicating to them that they cannot perform that action (through this App). However, it is possible that the App privileges have changed at the App Admin. Thus, the App may try getting a new appToken, and seeing in the privileges field if there is any change from its earlier appToken. If so, it may try getting a new accessToken based on this new appToken and try the request again.
not_authorized	User is not authorized for this operation/request.	App must show corresponding error message to the user. If there is a user_message in the response, App must show it to the user as well.

Appendix I Acknowledgements

On behalf of the cable industry and our member companies, CableLabs would like to thank the following organizations for their participation in the development of this document:

Oscar Marcia, Seetharama Rao Durbha, Stuart Hoggan and Dave Belt – CableLabs

IPRemote and CAFÉ project teams – CableLabs

Seetharama Rao Durbha, CableLabs Team Lead